Acceleration Amplitude Control System for Oscillating Silicon Oil Physics Experiment

By

Eric Chen

Senior Project

Electrical Engineering Department

California Polytechnic State University

San Luis Obispo

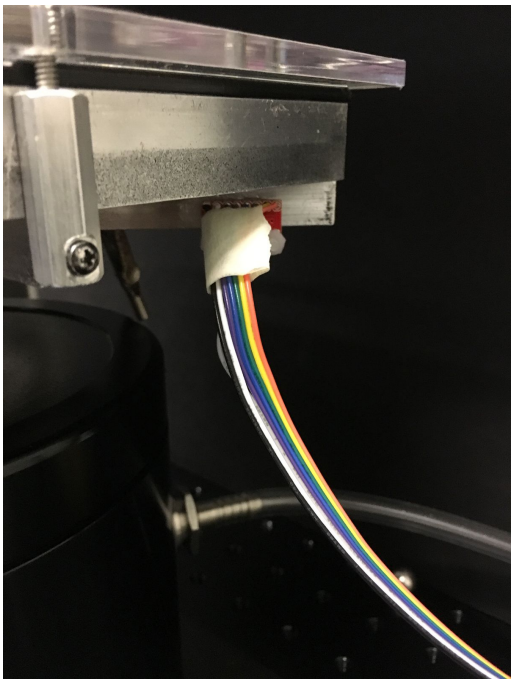
2017

Table of Contents

List of Tables and Figures

Abstract

This project is an ‘acceleration amplitude control system’ for ‘oscillating silicon oil physics experiment’. The physics experiments has a tray of silicon oil, this tray must oscillate vertically at a specific amplitude (either at or just below the level for Faraday Waves). This control systems manages taking measurements of acceleration amplitude, and also correcting the shaker’s driving signal to obtain the correct acceleration. Features of the control system are designed for ease of use, because the system will mostly be operated by professor/students without background knowledge in electrical engineering.



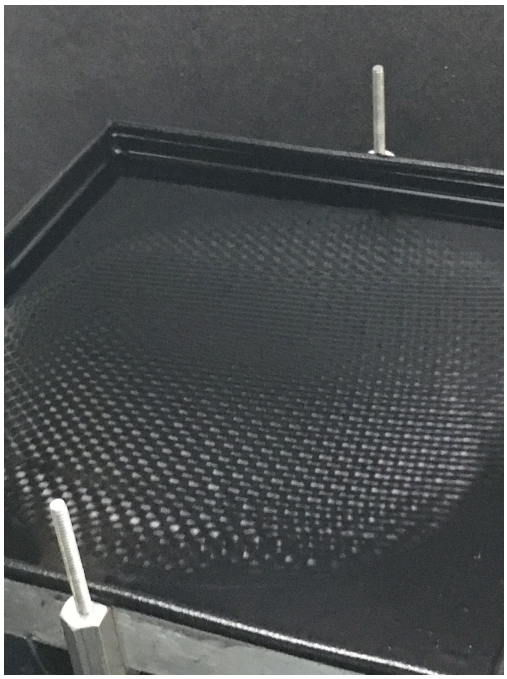


Figure 1 (Left): Accelerometer Attached to Tray
Figure 2 (Right): Tray of Silicon Oil, With Faraday Waves

Introduction

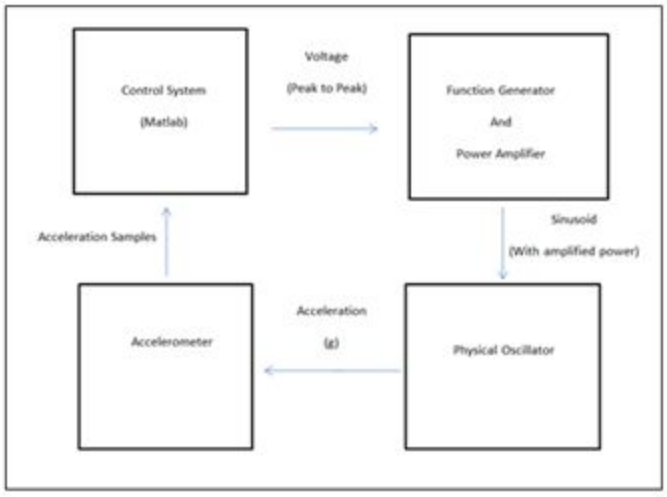


Figure 3: Block Diagram of Control System Version 1

This Senior Project is for an Oscillation Control System. The acceleration amplitude of a physical oscillator, which is being driven at a specific frequency (80Hz), is set by user input. The goal of this project is to quickly maintain the desired acceleration amplitude as accurately as possible while being easy to use and safe(no system failure that can damage equipment).

The control system uses error correction to adjust voltage of the function generator that drives the physical oscillator. A digital accelerometer is used to read the acceleration at fixed intervals (1kHz). This data is streamed to Matlab on a computer, where the control unit is located. There are failsafes to prevent instabilities caused by typos, disconnections, or other mistakes that were not considered.

As the system was being developed, issues such as noise and limited sampling rate became apparent. These issues are considered in the system design.

The hardware (arduino and accelerometer) has been packaged primarily for convenience. It is simple and straightforward to manage in order for non-EE’s to use it with minimal training. The software has been designed for this constraint as well.

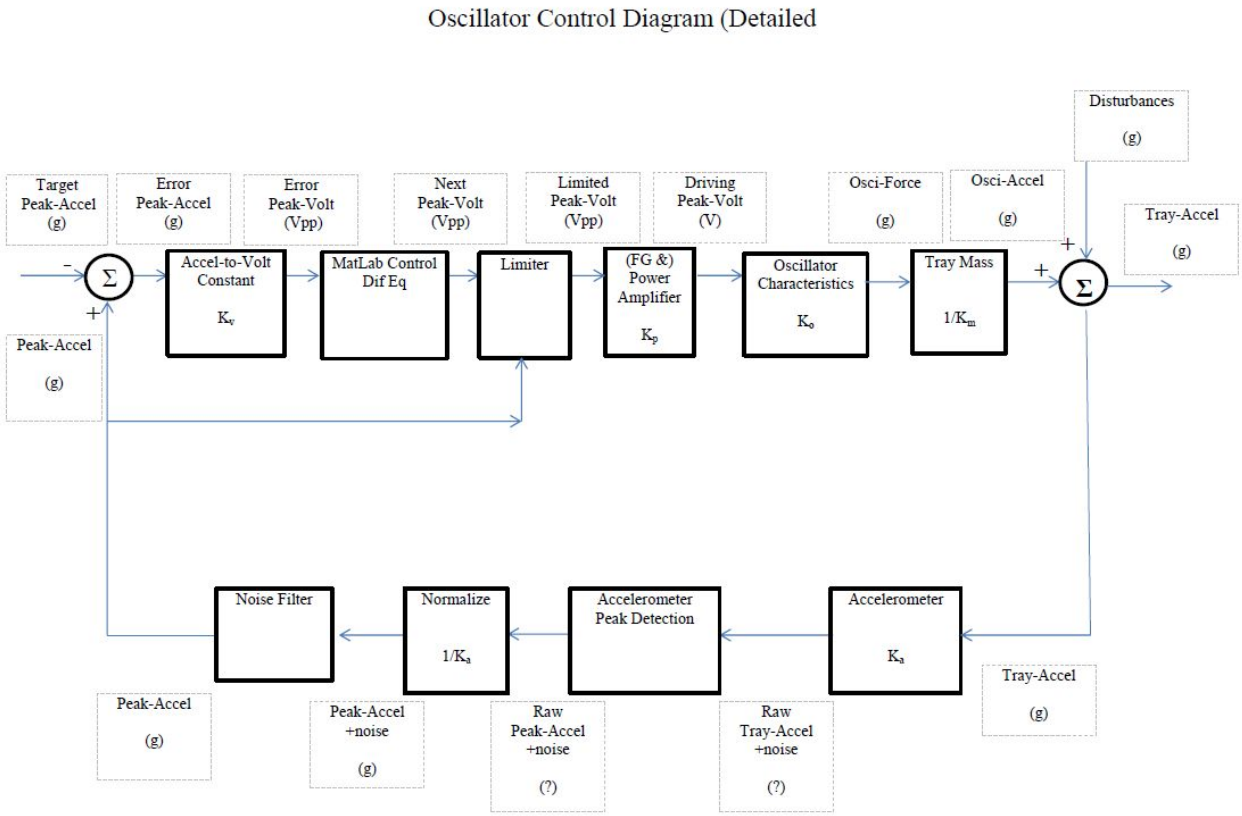


Figure 4: Control System Block Diagram

Accelerometer

The LIS331 digital accelerometer was chosen for its high resolution (0.003g / LSB), and decent sampling rate (1kHz). The communication protocol(SPI) was also a deciding factor because of its fast communication rate. From testing in winter quarter, the LIS331 is fairly consistent and accurate with measurements. However, there are noise spurs at +- 39Hz, which is likely caused by phase noise in the LIS331. This issue is being looked into in the Spring Quarter.

Arduino

A microcontroller was necessary as an intermediate device between the accelerometer and the computer (with MatLab). The arduino was chosen because it is robust, cheap, and has many libraries and examples available online for learning.

Matlab: Data collection and normalizing

The acceleration data is collected as a stream of raw, unnormalized samples of magnitude (instantaneous values, not the amplitude, which is the peak value). This is to minimize processing on the Arduino (which cannot handle divides very well), and allows the samples to be collected at the fastest possible rate (1kHz, matching the LIS331's maximum sampling rate).

The acceleration data was originally collected as maximums of consecutive 25 samples, which resulted in significant error because it did not account for noise. A later section discusses why this method was abandoned.

Matlab: Data processing

To obtain the maximum, the FFT is used in Matlab. This is in order to remove noise. The amplitude at the driving frequency is taken, ignoring spurs at other frequencies. This method is equivalent to putting the data through a perfect bandpass filter and then measuring the amplitude.

There is phase noise in the signal, likely caused by a PLL structure in the LIS331 accelerometer. Therefore a mathematical correction is applied to the measured amplitude, which was diminished by the periodic phase noise.

Matlab: Control

The desired amplitude is obtained by error correction with the measured amplitude. The original control system equation was a first order (integrate the error on the to function generator voltage). Because of the large time delayed introduced from the FFT (1 second window), this equation will need to be reviewed to prevent overshoot and minimize correction time.

As of today, the system is capable of reaching 0.1% steady state error in 5 seconds, and having less than 5% overshoot.

Matlab: Failsafes

There are multiple fail safes in case of disconnection, unforeseen instabilities, and typos. These failsafes were reviewed and tested thoroughly before proceeding with testing the main program.

Before each update of the function generator voltage, values are compared with limits. The outgoing voltage has a minimum and maximum, or else the update will be canceled and an error flag will be raised. The acceleration to voltage ratio has a minimum and maximum. This allows the program to determine if something could be wrong, and then cancel the program or proceed cautiously. If this ratio is too high or too low, it means the power amplifier setting could be wrong, or communication has been broken between the arduino and accelerometer.

If communication between Matlab and either the Arduino or Function generator is broken, the control program is terminated.

The user interface uses gated inputs. A new acceleration value is first typed in, and then must be confirmed by the a button press. This method is slower than ungated inputs, but is safer because it helps prevent user typos. The user is only expect to input new acceleration periodically (realistically once or twice for a full experiment), so the seconds lost are negligible. If the user inputs a target acceleration amplitude that is too high, then the program will reject this value and print an error message.

Matlab: Communication rate

The communication protocols are relatively limiting.

The 9600 baud rate with the function generator limits the voltage update rate to 10Hz. The selected update rate was set to 10Hz.

The 112500 baud rate with the arduino limits the acceleration read rate to 1.41 kHz. The selected read rate was 1 kHz, because is the highest possible for the accelerometer.

Matlab: User Inputs and User Feedback

The user may input acceleration amplitude, maximum/minimum voltage, and maximum/minimum acceleration.

The program reads in user inputs once per program cycle, at the beginning. This is to prevent possible instabilities from changing a value at a random program location.

The user feedback includes their inputs, target acceleration, function generator voltage, plots of data in time domain and frequency domain, and colored background to make errors obvious. These are updated only once every 500ms, because the updating of plots and gui figures takes too much time (~14ms) to occur at every cycle. For the rate a human can read and process the data updates, this rate seems sufficient.

Requirements

- Measurement of Acceleration Amplitude
  - Accuracy of 0.5%
- Setting of Acceleration Amplitude
  - Type Acceleration into GUI
  - From startup to reaching Faraday Threshold, take at most 20 seconds
- Setting Function Generator Voltage and Frequency
  - Type Voltage/Frequency into GUI
- User Friendliness
  - Easy to use GUI
  - Hardware and software easy to setup
- Safety
  - Shutdown if Acceleration above 7g
  - Shutdown if Voltage above 1100mVpp
- Stopwatch
  - Resolution 0.01s
  - Record a log of durations
    - Also record the corresponding acceleration, voltage, and frequency

Design

The designs are as described in the introduction

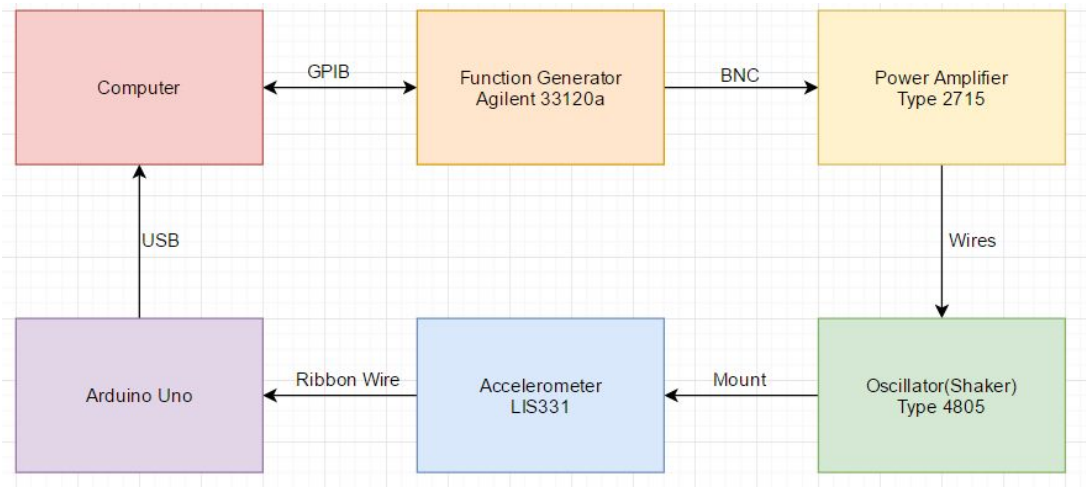


Figure 5: Block Diagram of Control System Version 2

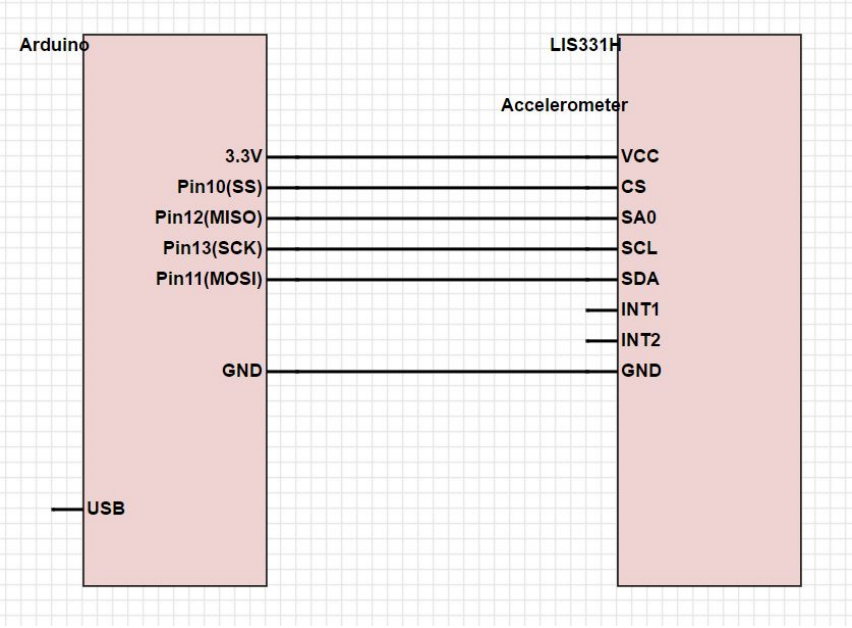


Figure 6: Wiring diagram for Arduino and LIS331 Accelerometer
SPI communication, interrupts unused.

Arduino is used between accelerometer and LIS331 to handle communications and timing. Arduino interrupts were fine tuned to be as close to 1kHz as possible (after callibration oscilloscope: within 0.0006% of 1kHz)

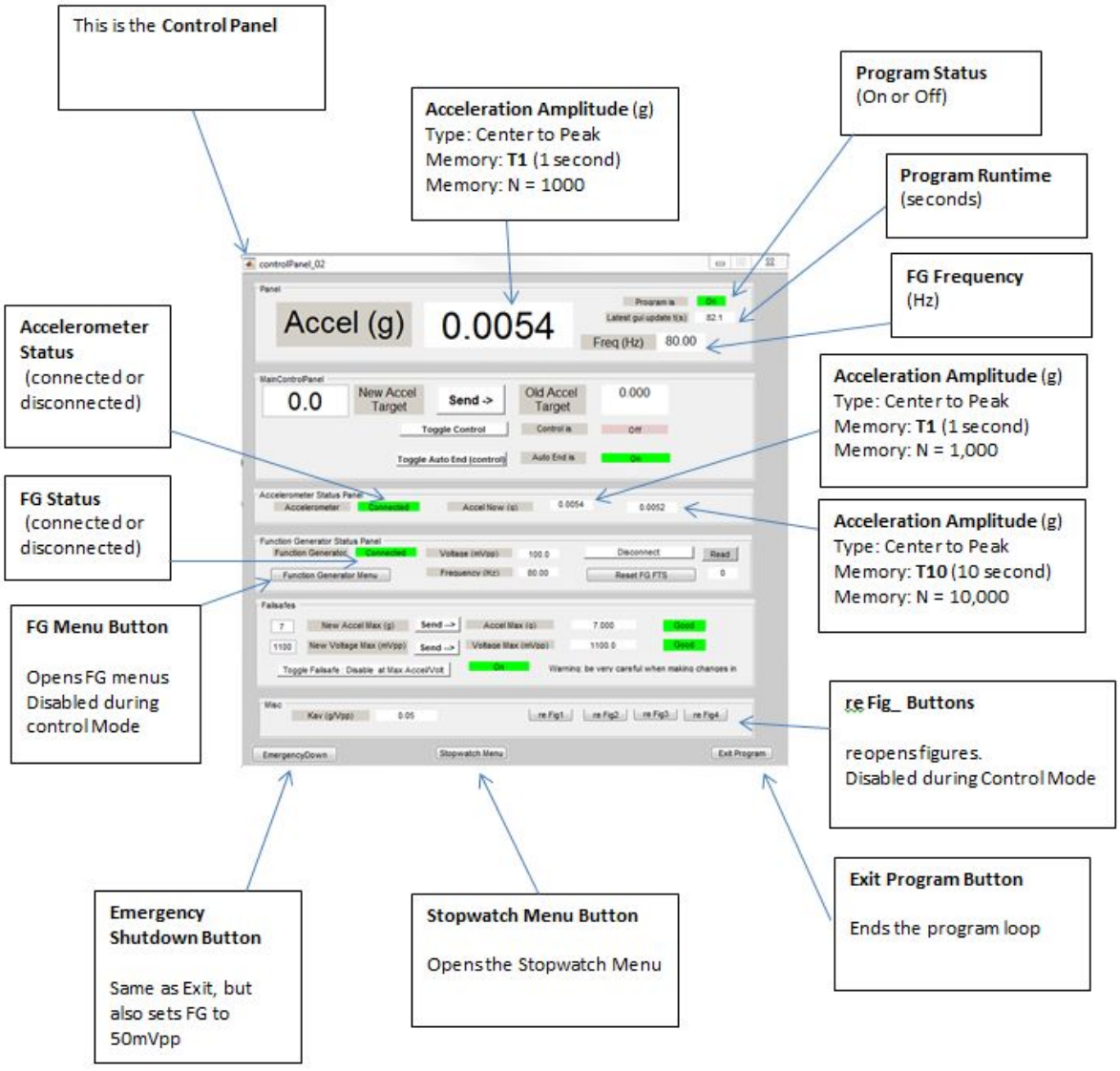


Figure 7: Control Panel, labeling 1

Function of the control panel

Display Acceleration	Display Frequency	Display Runtime
Set Acceleration	Control Mode Settings	Display Arduino Status
Display FG Status	Display FG Voltage	Display FG Frequency
Display Failsafes	Emergency Shutdown	Exit Program
Open other menus	Open figures	

Not all the elements are labeled on Figure 7, see Figure 8 for the rest.

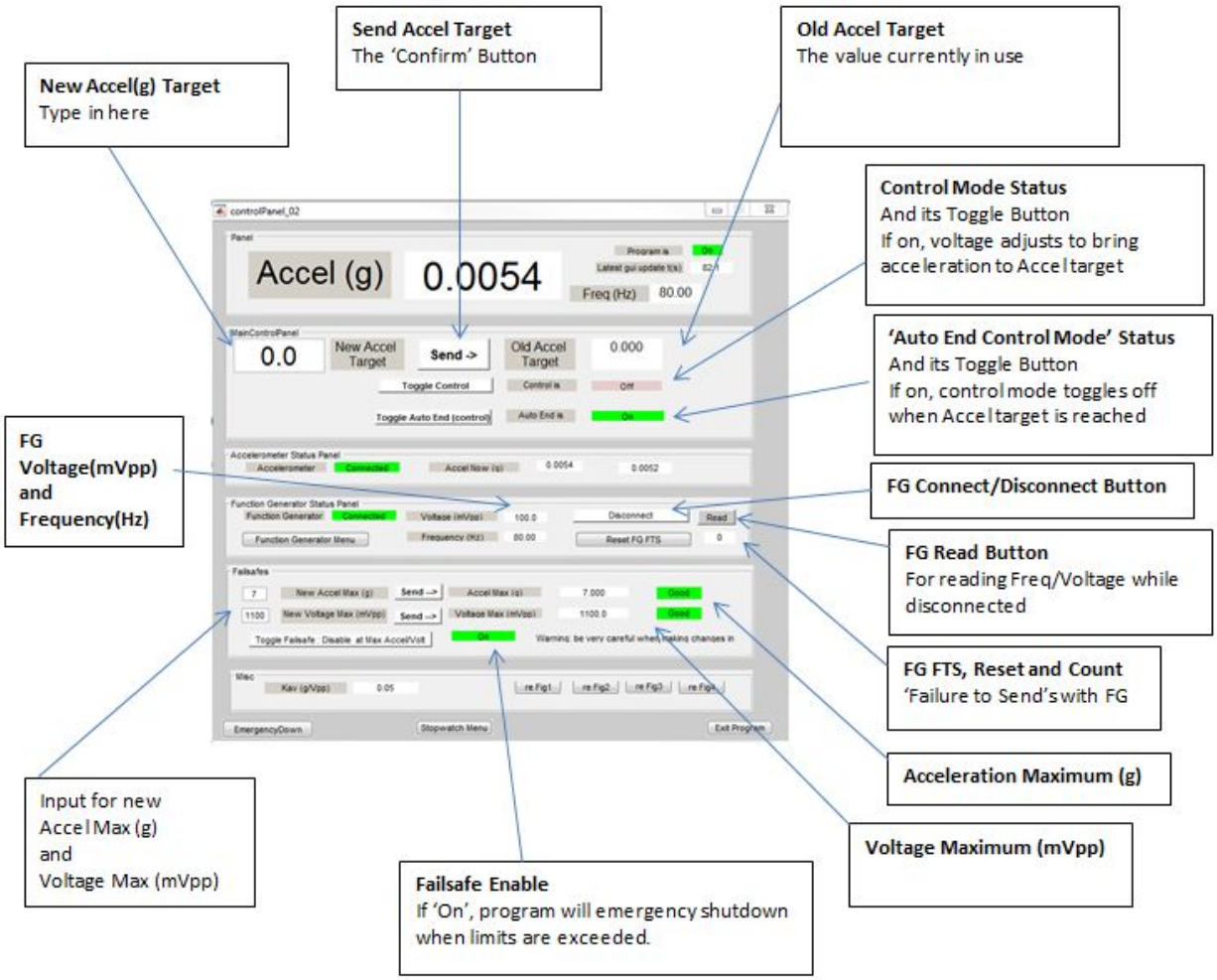


Figure 8: Control Panel, labeling 2

The rest of the buttons/displays/elements are labeled on this figure.

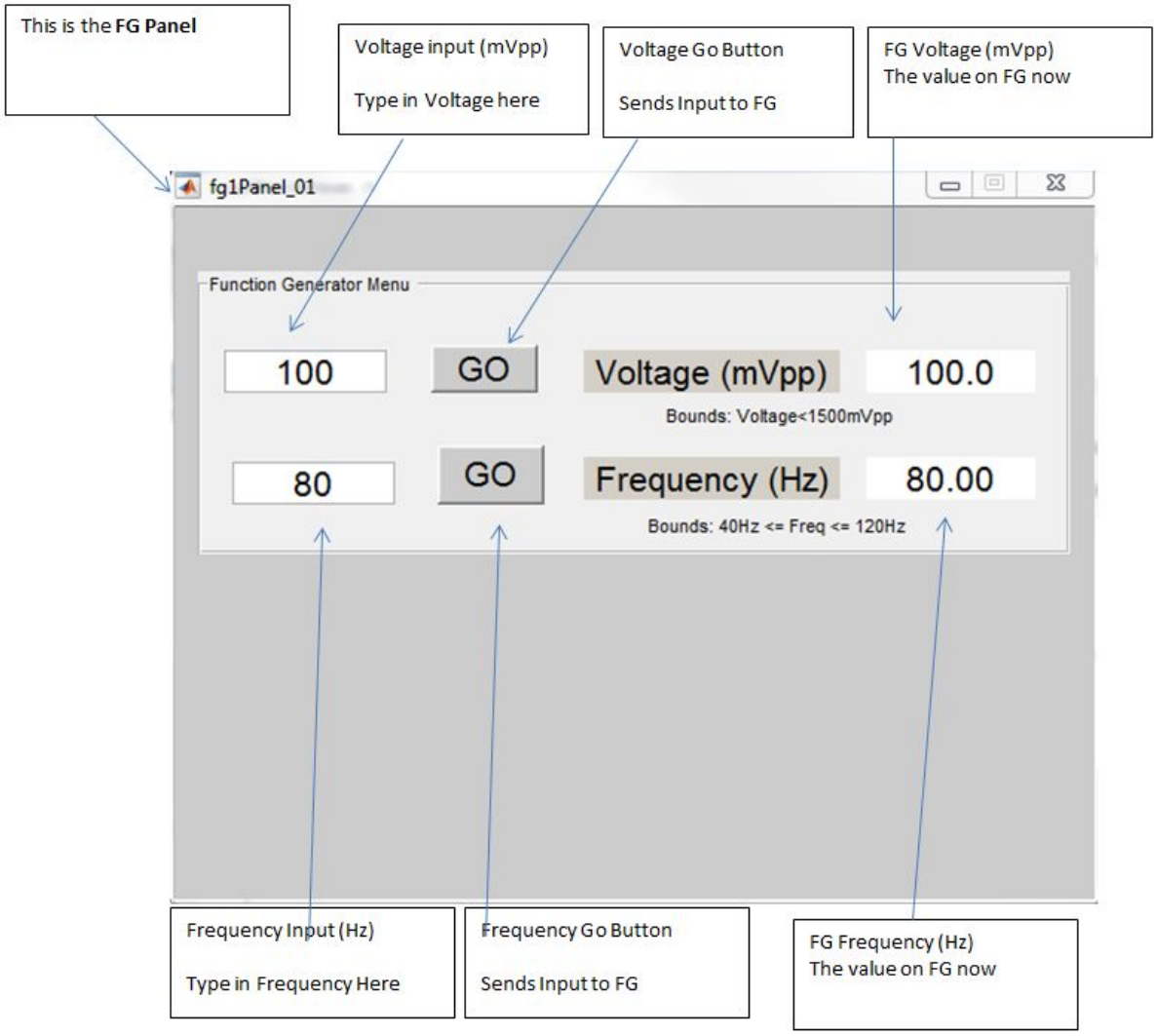


Figure 9: Function Generator Menu

The function generator is for setting the function generator's voltage and frequency (instead of setting acceleration).

The 'go' buttons are confirmation buttons. These act like speed bumps: the added inconvenience prevents typos and other mistakes from occurring when pressing the button. When confirming, the values are compared with safe values for voltage and frequency.

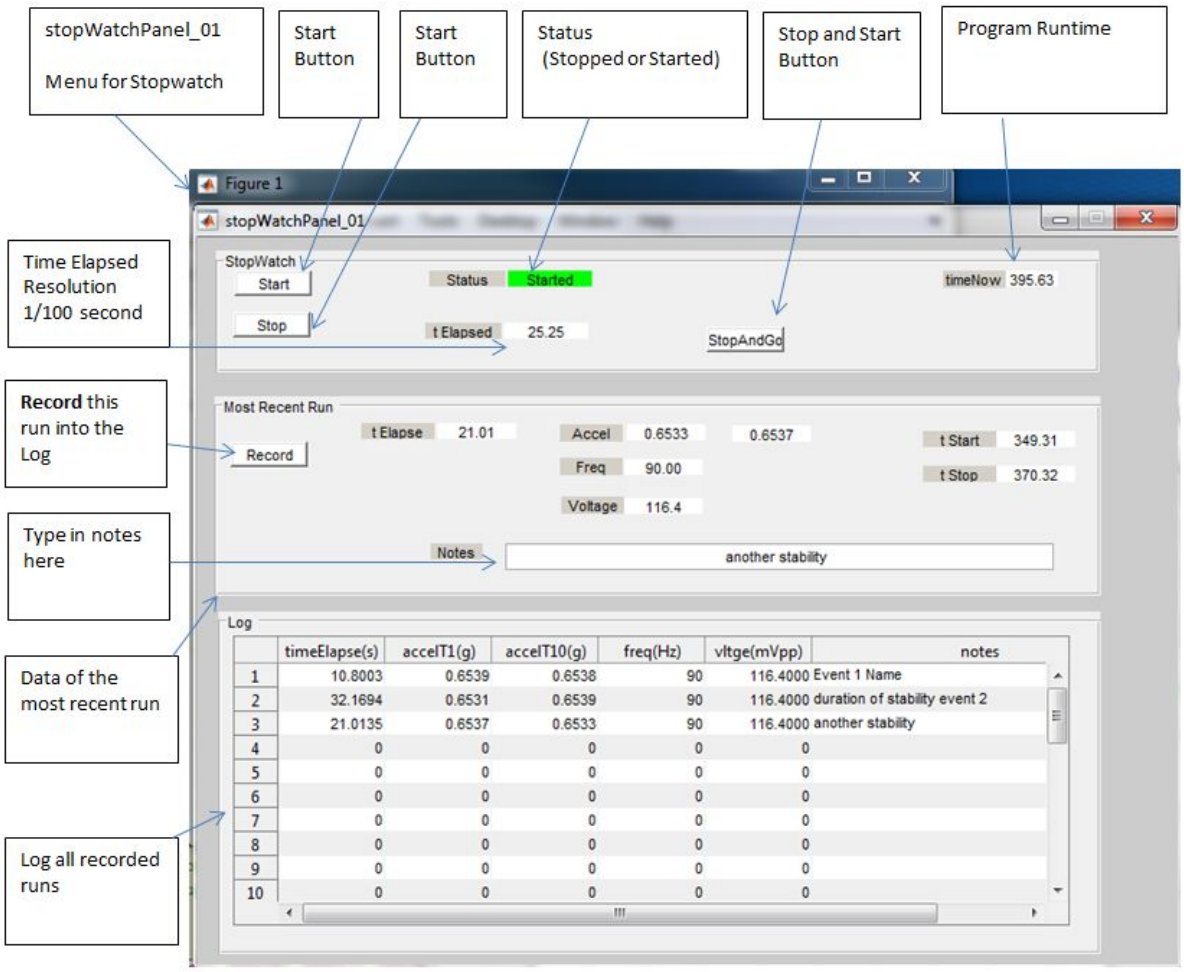


Figure 10: Stopwatch Menu

The stopwatch works as a normal stopwatch, with start and stop buttons.
The time recorded has resolution of 0.01 second.
Stopwatch times can be recorded to a log, which also records all the other information (acceleration, frequency, voltage).
After control system is ended, these are saved to both a matlab file and an excel file.

Testing

Test timing of Arduino Interrupts, using Oscilloscope observing CS pin
Validate data sampling rate with FFT

Measure the accelerometers precision

Characterize accelerometer's linear range

Measure accelerometers accuracy

Test functionality of Program
- Set function generator
- Failsafes

Testing Results

Timing-

Arduino interrupts were fine tuned to be as close to 1kHz as possible (after callibration oscilloscope: within 0.0006% of 1kHz)

Sampling Rate-

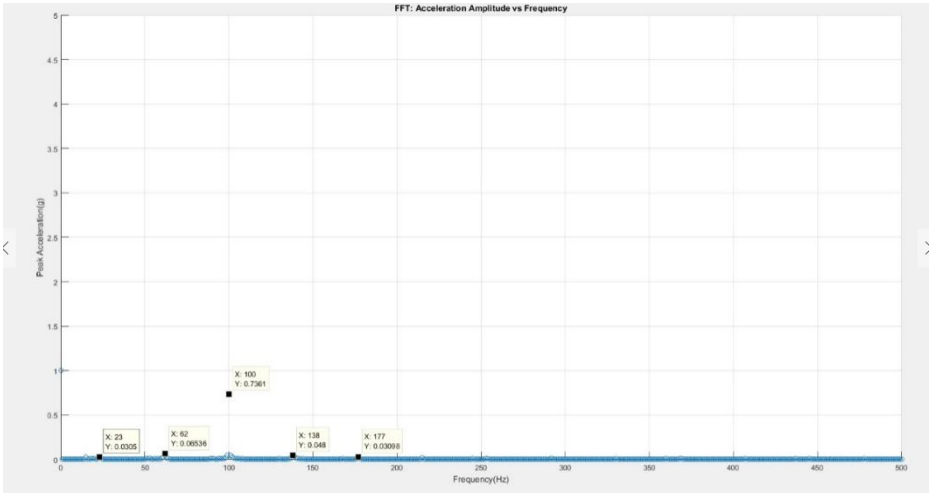


Figure 11: FFT of data
80 Hz Oscilliation appears at 80Hz, as expected.

Precision-

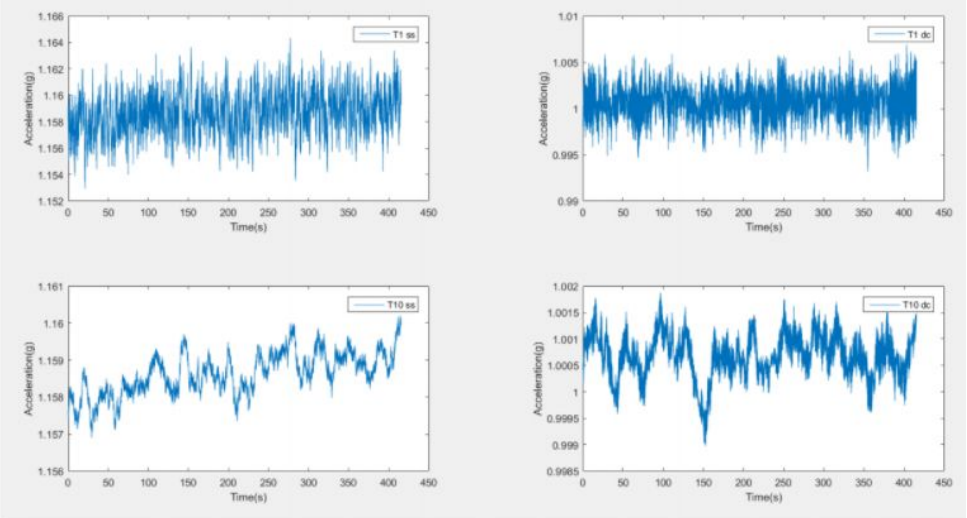
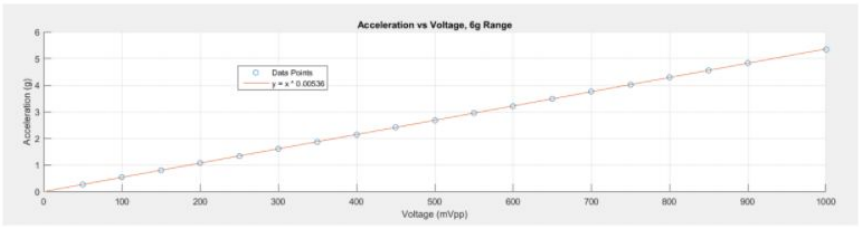


Figure 12: Data recordings of 6.8 minutes (16k data points)
Used these for determining precision

T1 Acceleration Data Uncertainty: 0.45%

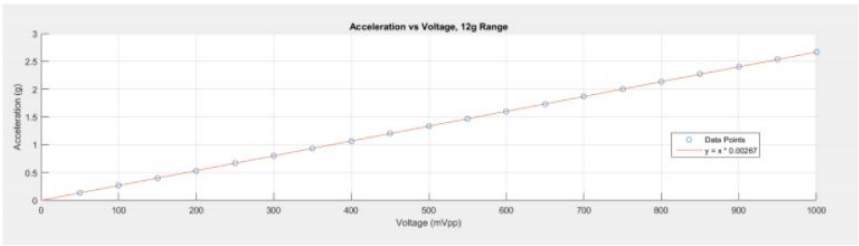
T10 Acceleration Data Uncertainty: 0.15%

Linear Range-



Using T1 Data, precision approximately 0.45%

Max Deviation: 0.36%



Using T1 Data, precision approximately 0.45%

Max Deviation: 0.4%

Figure 13: Characteristic Plot, Acceleration Amplitude vs Voltage

Result: Accelerometer is linear up to at least 5.5g (center to peak).
This is beyond the required ranged.

Accuracy-

The calibration methods of this accelerometer are potentially not sufficient.

Measure Acceleration of mounted accelerometer, at 0Hz.

Find the 1g value

Measure Acceleration of mounted accelerometer at 0Hz, upside down

Finds the -1g value

These are used to determine

Calibration constant y1g = 2642

Calibration constant y0g = -236

The calibration was compared to the expected Faraday Level for this specific viscosity Silicon oil.

Expected:	4.2g
Measured:	4.24g
Error:	1%

This error in Faraday level could be from error in calibration, or a difference in characteristics of this silicon oil tray system. It is to be determined.

Functions

Set function generator voltage and frequency

These function worked as expected, and consistently

Failsafes:

Various test cases were used, the emergency shutdown was triggered in all cases, consistently.

Voltage too high	Successful Emergency Shutdown
Acceleration too high	Successful Emergency Shutdown
USB Fail to Communicate	Successful Emergency Shutdown
GPIB Fail to Communicate	Successful Emergency Shutdown
Accelerometer unplugged	Successful Emergency Shutdown

Testing by Intended User

The program was tested by people without knowledge of the system design. The user interfaces and user manuals were sufficient. The program was deemed sufficiently user friendly.

Conclusion

All the functions required were implemented and tested. The control system measures acceleration and sets it by user input, by changing the driving signal for the shaker. The secondary functions were implemented as well (emergency shutdown, set voltage, set frequency, stopwatch, record). Accelerometer was determined to have sufficient linear range.

The precision and accuracy are slightly questionable (0.15% and 1%), but these are being worked on. A new accelerometer will eventually replace the cheap low quality one currently in use. For now, acceleration measurement can reliable be used for relative measurements, but precision acceleration measurements will have to wait until the new accelerometer is added.

Appendix A) Senior Project Analysis

Project Title
Acceleration Amplitude Control System for use in OScillating Silicon Oil Physics Experiment

Advisor(s)
Advisor: Professor Xiaomin Jin
Unofficial Advisor: Professor Nilgun Sungar

Functional Requirements
Accurately measure acceleration amplitude of a sinusoidal oscillation. Using this measurement as feedback, use measurements to set driving signal amplitude, in order to set acceleration amplitude to a desired value.

Primary Constraints
Communication rates, signal-noise-ratio of accelerometer, resolution of accelerometer, cost of accelerometer, and processing speed of computer.

Economic
Economic impacts expected: none, this is a relatively small scale project.

If manufactured on a commercial basis
Not manufactured on a commercial basis

Environmental
Environmental impacts expected: none, this is a relatively small scale project.

Sustainability
The parts used (IC's, test equipment, breakout boards) are not renewable. However, the amount used is negligible, therefore making this project sustainable.

Ethical
There are no unethical uses for this project.

Health and Safety
There are no health and safety concerns associated with this project.

Social and Political
There are no social and political issues associated with this project.

Appendix B - Specifications and Requirements

Table 13: Specifications and Requirements

Measurement of Acceleration Amplitude	Accuracy of 1%
Setting of Acceleration Amplitude	Type Acceleration into GUI From startup to reaching Faraday Threshold, take at most 20 seconds
Setting Function Generator Voltage and Frequency	Type Voltage/Frequency into GUI
User Friendliness	Easy to use GUI Hardware and software easy to setup
Safety	Shutdown if Acceleration above 7g Shutdown if Voltage above 1100mVpp
Stopwatch	Resolution 0.01s Record a log of durations Record the acceleration, voltage, and frequency

Appendix C - Parts List and Costs

Table 14: Parts List and Costs

Part	Cost ($)	Quantity	Total($)
LIS331 Accelerometer	12.5	2	25
Arduino Uno	25	1	25
Ribbon Wire (3m)	7	1	7
Black Project Box	6	1	6
Nylon Screws/Nuts	1	1	1
Total			64

Appendix D - Schedule - Time Estimates

Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W10	W12
MONTH	Jan				Feb				Mar			
(MON)DAY	9	16	23	30	6	13	20	27	6	13	20	27
Acclmeter Wiring												
Arduino Program												
Control Panel												
FG Panel												
First Design Review												
First Half Demonstration												
1st draft of report												
Weekly advising report												
User manual												

Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W10	W12
MONTH	April				May					June		
(MON)DAY	3	10	17	24	1	8	15	22	29	5	12	19
Stopwatch Panel												
Failsafe testing												
Program commenting												
Program testing												
Full system testing												
Weekly advising report												
User manual												
Demo and Report												
Project Expo(June 2)												

Figure 15: Gantt Charts

Appendix E - Program Listing

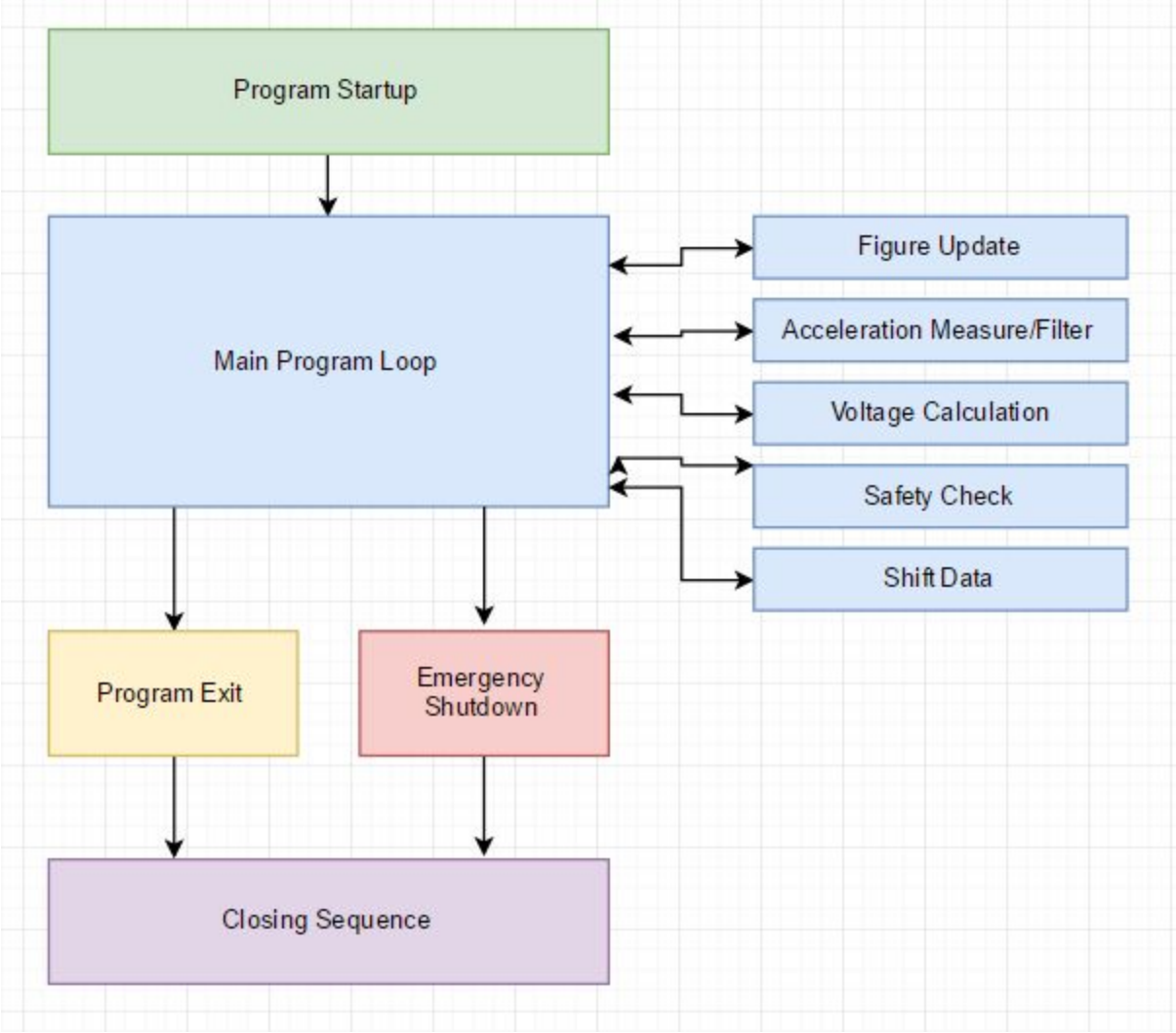


Figure 16: Program Flowchart
This is the flowchart for the MatLab program

Appendix F - Hardware Configuration/Layout

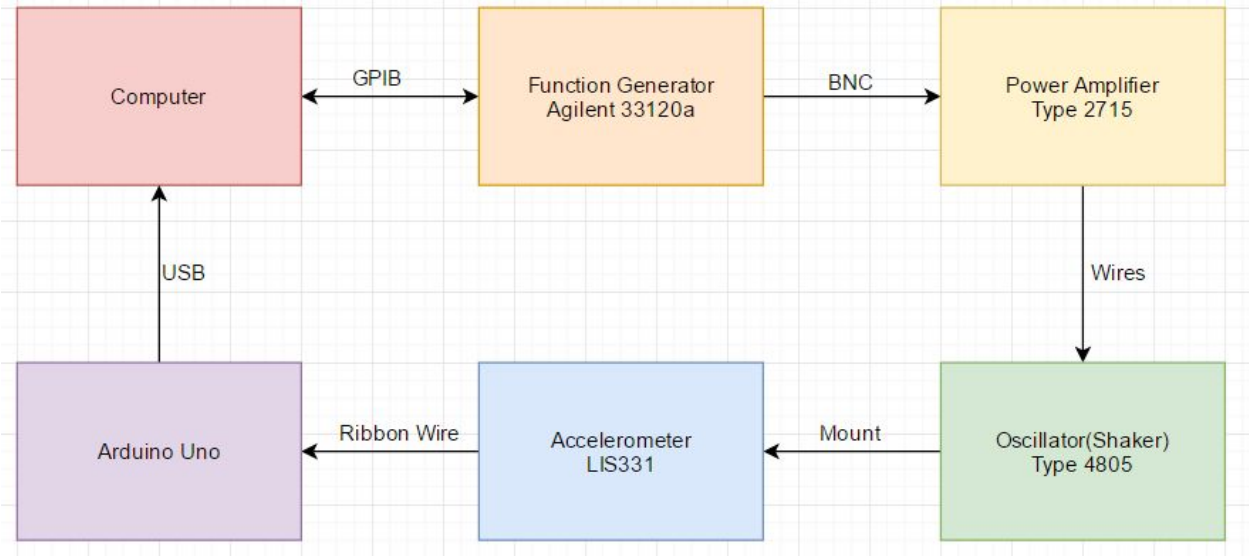


Figure 17: Block Diagram of Control System Version 2

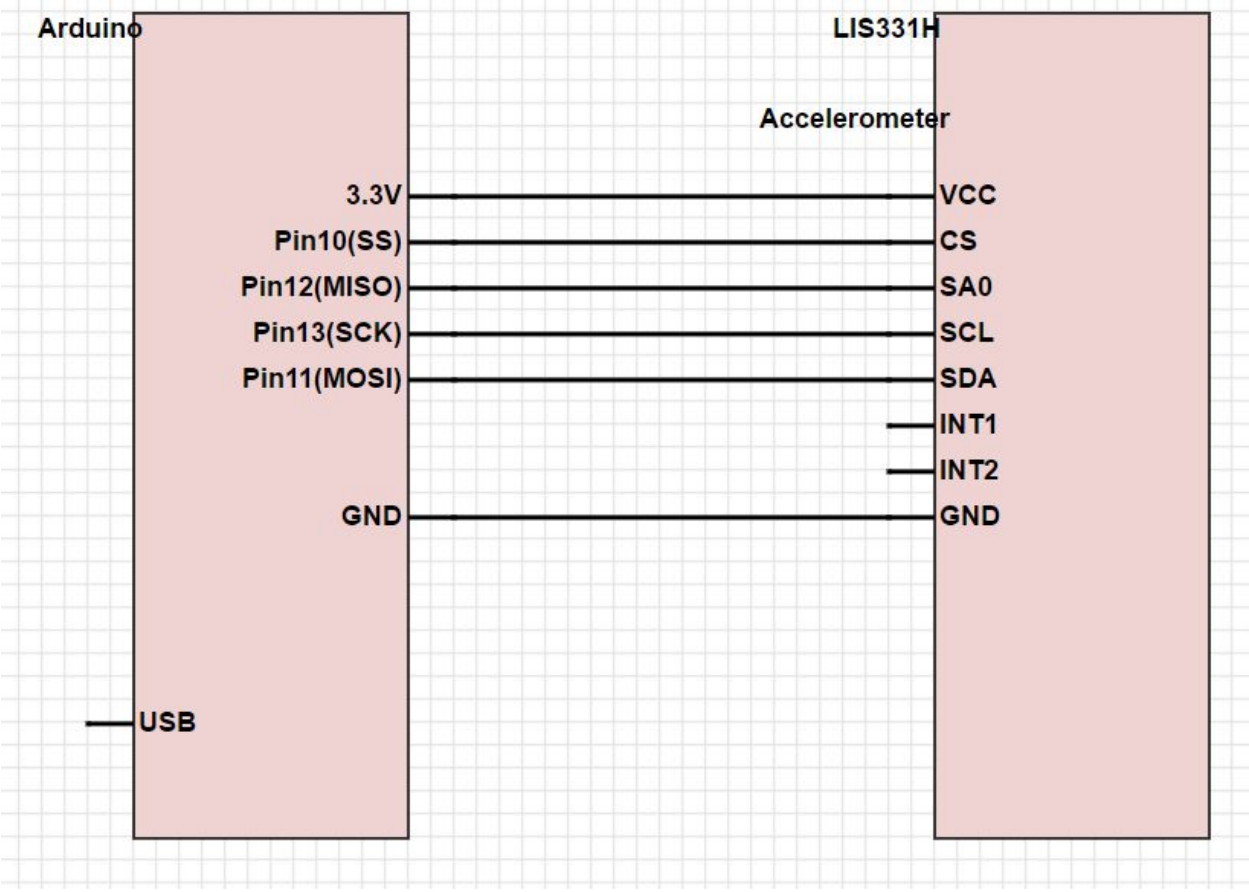


Figure 18 Wiring diagram for Arduino and LIS331 Accelerometer

Appendix G - Software

The code included in this report are only part of the program. Many other subroutines are included in a zip file with the entire program.

m_main.m This is all the program set up

```
%%
% V04, 2017/05/30T
% Step By Step Checkpoint
%%

clear all;
delete(instrfindall);
%%
%To be Commented Later -Eric
addpath('add_scripts01');
addpath('add_functions01');
addpath('add_scripts02');
addpath('add_scripts03');
addpath('guis');

fprintf('\nBeginning initializations\n');
f_settings; %select operations to run, and other high level details

          %Yes, these numbers are out of order on purpose
f_init1;    %initialize variables, and other low level stuff
f_init6;    %initialize timestamp
f_init7;    %initialize flags and counts
f_init8;

if( rec.enable == 1)
   addpath('add_scripts04r');
   init_r01;
end;
% f_init2a;   %setup figure 2
%init 2a is disabled for now because it is relatively useless
f_init2b;   %setup figure 1, figure 3, figure 4

timeStamp;
fprintf('Settings, Init1, Init6, Init7, Init2a, Init2b Done\n');
drawnow;
```

```
%%
timeStamp;
fprintf('Beginning init3a, ard1 initializations\n');
f_init3a;    %initialize communications (arduino1) and plots
        %try/catch and flags included inside of function
timeStamp;
fprintf('init3a complete\n');

timeStamp;
fprintf('Beginning init3b, fg1 initialization. If freeze here, FG might be plugged in but off.\n');
drawnow;
f_init3b;    %initialize functionGen1
        %try/catch and flags included inside of function
fprintf('init3b complete\n');

% timeStamp;
% fprintf('Init3a, Init3b\n');

%% gui data
f_init4;    %gui data
timeStamp;
fprintf('Init4 Done\n')
timeStamp;

fprintf('control system initializations completed\n');
%%
fg1Menu = fg1Menu_01;
handlesfg1Menu = guidata(fg1Menu);

%%
controlGui = controlPanel_02;
handles = guidata(controlGui);

%%
timeStamp;
fprintf('control system gui data loaded\n');
f_init5; %some more gui data. must occur after gui is opened
timeStamp;
fprintf('Init5\n\n');
timeStamp;
fprintf('Program is now looping\n\n');
f_updateToGUI;%Pre update
f_secondary;    %This is the loop
```

F_secondary.m This is the program loop

```
guiUpdate.Previous = toc;
guiUpdate.Since = toc - guiUpdate.Previous;
vSendError.Previous = toc;
vSendError.Since = toc - vSendError.Previous;

%% test 10 samples on arduino1 read
fprintf('\n')
timeStamp;
fprintf('testing 10 reads arduino1 [');
for i = 1:10;
      try
          this = fscanf(arduino1, '%f');
          fprintf('%.0fy_',i);
      catch
         fprintf('%.0fn_',i);
      end;
end;
fprintf('] tested 10 reads arduino1 \n');
f_updateFromGUI;
cps_centralProcessingScript;
%%
% while(guiData.programIsOpen == 1 && flags.programEmergencyDown == 0 &&
flags.FTC_functionGen1 == 0 && flags.FTC_arduino1 == 0);
while(flag.enable_run == 1);
      f_updateFromGUI;

      %guiUpdate only every 0.5s
      guiUpdate.Since = toc - guiUpdate.Previous;
      if(guiUpdate.Since > 0.5)
         f_updateToGUI;  %
         guiUpdate.Previous = toc;
      end;

      stopWatchRoutine;

      %print a status message every 5
      periodicPrint.Since = toc - periodicPrint.Previous;
      if(periodicPrint.Since > 5)
         fprintf('\n');
```

```
        timeStamp;
        fprintf('Osci Control System Program is Running\n\n');

        if( rec.enable == 1)
            f_recPrint;% for recordingh
        end;

        periodicPrint.Previous = toc;
    end;



    %%
    if(flag.enable_run == 1 ) %originally was flag for data reading. but data reading is always
            f_getAccel; %read accel
                    %if fail to connect will detect and set flag.
                    %records failures per set of 25

            if( rec.enable == 1)
                if(guiData.autoEndMode == 0 && 1 == 1)
                    f_recAccel;
                end;
            end;


            f_dataCalc;

    end



    %%
    if(flag.enable_voltControl == 1)

        f_getNewVolt02;   %calculate new Voltage to send
%           [yAccel, fg1, tempfg1Volt, counts ] = f_getNewVolt02( dataSettings, n, datas, yAccel,
fg1, counts);
        f_voltLimit;    %apply limits
%           [fg1.Volt, fg1.failSafeCount, vSendError] = f_voltLimit(tempfg1Volt, fg1,
fg1.failSafeCount, vSendError);

        f_checkMaxAccelVolt; %if beyond limits, will set emergency down, and program will end
```

```
        end;
        f_checkEmergencyDown;
        %% goes right before voltSend
        cps_centralProcessingScript;

        %%
        if(flag.enable_voltSend == 1)
            f_voltSend;
        end;
        %%
        if(flag.enable_fg1MenuRoutine == 1);
            f_fg1Menu_routine;
        end;
        %%
        if(flag.enable_figUpdate == 1)
            f_plotData;     %plot data for visuals
            f_figReopen;
        end;                %last because it is slowest


    if(flag.enable_toggleFunctionGen == 1)
        f_toggleFunctionGen1;
    end;

    if(flag.enable_reReadFG1 == 1)
        f_reReadFG1;
    end;
end;
%%
% if(modes.record == 1);
%     [recorded_seg] = f_record(800, arduino1, y1g, y0g);
% end;
%%
f_exitSequence;     %exit it the program.
```